Teórico 4
SQL (Primera parte)

Structured Query Language

SQL

- SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos relacional, "SQL" es una abreviatura de Structured Query Language (Lenguaje de Consultas Estructurado).

- Veremos el estándar SQL92, y algunas características del SQL99, la mayoría de los manejadores de bases de datos (DBMS) actuales implementan SQL99, pero generalmente cada implementación puede tener algunas variantes al estándar.

Historia del Estándar

- El Lenguaje SQL fue diseñado sobre un prototipo de IBM que incluía un lenguaje de consulta a base de datos llamado SEQUEL.

- El Instituto de estandarización ANSI y más tarde la ISO adoptan SQL como el lenguaje estándar de consulta a bases de datos.

- SQL 2006 incorpora la interacción con XML.

- En la actualizad el estándar es SQL2008.

- Aunque el ANSI/ISO SQL es el estándar, ORACLE presentó su propia versión comercial, y más tarde lo han hecho Microsoft (SQL SERVER) y otras empresas. En la actualidad MySQL es de libre distribución y su servidor es el más utilizado en servidores de bases de datos de Internet.

DML, DDL y DCL

- SQL está dividido en tres partes:
  - Lenguaje de Manipulación de Datos (DML siglas en Ingles): Brinda instrucciones que incluyen un lenguaje de consultas, basado en el Álgebra Relacional, además incluye de sentencias para agregar modificar y eliminar datos.
  - Lenguaje de Definición de Datos (DDL siglas en Ingles): Proporciona instrucciones para la definición del esquema (Estructura) de la Base de Datos, por ej.: crear tablas, índices, etc.
  - Lenguaje de Control de Datos (DCL siglas en Ingles): contiene instrucciones que permiten manipular los permisos sobre las bases de datos y sus objetos.

DML

Comando SELECT

El comando más usado en SQL es la instrucción SELECT, se utiliza para recuperar datos de diferentes tablas de una base de datos. Consta de seis cláusulas: las dos primeras (SELECT y FROM) obligatorias y las cuatro restantes opcionales.

Su sintaxis es:

```
SELECT [ALL|DISTINCT]
        { * | expr_columna_1 [AS c_alias_1]
          [, expr_columna_2 [AS c_alias_2][,...]]}
  FROM nombre_tabla_1 [t_alias_1][, table_name_n [t_alias_n][,...]]
  [WHERE condicionWhere]
  [GROUP BY expr_columna_group1  [,expr_columna_group2] [,...]]
  [HAVING condicionHaving]]
  [{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
  [ORDER BY nombre_campo_i1 [ASC|DESC]
                  [, nombre_campo_j1 [ASC|DESC]][,...]]];
```

Cláusula SELECT

La cláusula SELECT lista los datos a recuperar por la sentencia SELECT. Los elementos o datos a seleccionar pueden ser columnas de la base de datos o columnas a calcular por SQL cuando efectúa la consulta. Su sintaxis es.

SELECT [ALL|DISTINCT]
{ * | *expr_columna_1* [AS *c_alias_1*]
[, *expr_columna_2* [AS *c_alias_2*][,...]]}

Argumento	Descripción
ALL	Si hay registros repetidos en la consulta también son devueltos.
DISTINCT	Si hay registros repetidos en la consulta, sólo devuelve uno de los repetidos.
*	La tabla resultante de la consulta tendrá todas las columnas de las tablas involucradas.
expr_columna_N	Define una columna, que será parte del resultado de la consulta, puede ser el nombre de una columna de las tablas participantes en la consulta, o una expresión que la involucre (por Ej.: Precio * 2). También puede ser una función agregada de SQL (ver funciones agregadas).
c_alias_N	Nombres que se van a utilizar como encabezados de columnas en la tabla resultante, en vez de los nombres de las expresiones de columnas.

Cláusula FROM

La cláusula FROM lista las tablas que contienen los datos a recuperar por la consulta. La sintaxis de esta cláusula es:

FROM *nombre_tabla_1* [*t_alias_1*][, *nombre_tabla _2* [*t_alias_2*][,...]]

Argumento	**Descripción**
nombre_tabla_N	Nombre de una tabla que contiene los datos que desea recuperar.
t_alias_N	Nombre que se usa para referirse a la tabla en el resto de la sentencia SELECT para abreviar el nombre original y hacerlo más manejable, en el caso de existir más de una tabla en la consulta, y también, para poder realizar consultas uniendo varias veces la misma tabla.

Ejemplos

De ahora en mas, los ejemplos de consultas SQL estarán hechos sobre el siguiente esquema de una base de datos.

articulos(Nart, desc, precio, cant, stock_min, stock_max)

Provee(Nprov,Nart, precio_venta)

Proveedores(Nprov, nombre, direccion)

Clientes(Ncli, nombre, direccion)

Compran(Ncli, Nart)

Con las siguientes instancias:

articulos

Nart	descr	precio	cant	stock-min	stock-max
100	Café	5.2	8	10	50
101	Azucar	1.2	30	15	45
102	Harina	1.1	35	16	40

proveedores

Nprov	nombre	dir
200	Grasano	Bs As 100
201	Atomo	Lima 85
202	VEA	Alvear 455

provee

Nprov	Nart	precio_Venta
200	100	4
200	101	1.1
201	101	1
201	102	0.5
202	102	1

clientes

Ncli	nombre	dir
300	Pedro	Bs As 90
301	Juan	Jujuy 800
302	Maria	Cordoba 24

compran

Ncli	Nart
300	100
300	102
301	101
302	101

Ejemplos

1) Devolver un listado de artículos con el número, descripción y precio de cada artículo.

```
SELECT Nart, descr, precio
FROM articulos;
```

Resultado:

Nart	descr	precio
100	Café	5.2
101	Azucar	1.2
102	Harina	1.1

Ejemplos (sigue)

2) Devolver un listado de artículos con el número, descripción, precio de cada artículo y la ganancia (es el 20% del precio del articulo) .

```
SELECT Nart, Descr, precio, precio*0.2 AS ganancia
FROM articulos;
```

Resultado:

Nart	descr	precio	ganacia
100	Café	5.2	1.04
101	Azucar	1.2	0.24
102	Harina	1.1	0.22

Ejemplos (sigue)

3) SELECT Nart FROM provee

Resultado

Nart
100
101
101
102
102

4) SELECT DISTINCT Nart FROM provee

Resultado

Nart
100
101
102

Ejemplos (sigue)

5) Devolver un listado con donde se combinen todos los proveedores con todos los precios de ventas de los artículos.

SELECT p.Nprov, Nart, precio_venta, pro.Nprov, nombre, dir FROM provee p, proveedores pro

p.Nprov	Nart	precio_Venta	pro.Nprov	nombre	dir
200	100	4	200	Grasano	Bs As 100
200	100	4	201	Atomo	Lima 85
200	100	4	202	VEA	Alvear 455
200	101	1,1	200	Grasano	Bs As 100
200	101	1,1	201	Atomo	Lima 85
200	101	1,1	202	VEA	Alvear 455
201	101	1	200	Grasano	Bs As 100
201	101	1	201	Atomo	Lima 85
201	101	1	202	VEA	Alvear 455
201	102	0,5	200	Grasano	Bs As 100
201	102	0,5	201	Atomo	Lima 85
201	102	0,5	202	VEA	Alvear 455
202	102	1	200	Grasano	Bs As 100
202	102	1	201	Atomo	Lima 85
202	102	1	202	VEA	Alvear 455

Cláusula WHERE

En cláusula WHERE se define la condición que deben cumplir las filas o registros de datos que estarán en el resultado de la consulta.

Su sintaxis es:

WHERE *condicionWhere*

Argumento	**Descripción**
CondicionWhere	Expresión que determina qué registros se incluirán en el resultado. Más adelante se explicará en detalle como se pueden construir condiciones.

Ejemplo

1) Devolver un listado con todos los campos de los artículos a reponer.

SELECT * FROM articulos WHERE stock_min >cant

Resultado:

Nart	descr	precio	cant	stock-min	stock-max
100	Café	5.2	8	10	50

Ejemplos (sigue)

2) Dar un listado con los proveedores y los artículos que provee con su precio de venta

```
SELECT Nart, precio_venta,p.Nprov, nombre, dir
FROM provee p, proveedores pro WHERE p.Nprov = pro.Nprov
```

Resultado

Nart	precio_Venta	Nprov	nombre	dir
100	4	200	Grasano	Bs As 100
101	1.1	200	Grasano	Bs As 100
101	1	201	Atomo	Lima 85
102	0.5	201	Atomo	Lima 85
102	1	202	VEA	Alvear 455

Expresiones

En una expresión se pueden cotejar con operadores de comparación, columnas con otras columnas o con expresiones del tipo de la columna, las expresiones se pueden conectar con otras por medio de operadores lógicos. Se pueden definir expresiones aritméticas, de cadenas, etc., dentro de las expresiones.

Operadores Lógicos

- **AND**
- **OR**
- **NOT**
- **XOR**

Operadores de Comparación

- **< , >, <>, <=, >=, =**
- **BETWEEN:** Utilizado para especificar un intervalo de valores. Utilizado en tipos String, numéricos y de fechas.
- **LIKE:** Utilizado en la comparación de un patrón. El carácter '%' se utiliza como comodín
- **In:** Utilizado para saber si el valor de un campo se encuentra en una subconsulta de selección de una columna.
- **EXIST:** Devuelve verdadero si una subconsulta **no** es vacía, si no, devuelve falso.

Ejemplos

a) Listados de artículos cuya descripción empieza con la letra A

SELECT * FROM articulos WHERE desc like "A%"

b) Listados de artículos cuya descripción empieza con la letra A y termina con la letra K

SELECT * FROM articulos WHERE desc like "A%" AND desc like “%K"

c) Devolver los artículos que son provistos por algún proveedor.

SELECT * FROM articulos WHERE Nart IN (SELECT Nart FROM provee);

d) Devolver los artículos que vendiendo 2 unidades mas estarían bajos en Stock

SELECT * FROM articulos WHERE stock_min >cant-2

Cláusula ORDER BY

La cláusula ORDER BY ordena los resultados de la consulta en base a los datos de una o más columnas. Si se omite, los resultados serán ordenados por el primer campo que sea clave en el índice que se haya utilizado.

Su sintaxis es:

ORDER BY *expresión_orden1* [ASC|DESC]

[, *expresión_orden2* [ASC|DESC]][,...];

Argumento	**Descripción**
Expresión_ordenN	Puede ser el nombre de un campo, expresión o el número de posición que ocupa la expresión de columna en la cláusula SELECT.
ASC	Produce el ordenamiento ascendente de los valores de la columna. Esta opción es la elegida por defecto.
DESC	Produce el ordenamiento descendente de los valores de la columna.

Ejemplo

1) Dar un listado con los artículos, que proveedores lo suministran y su respectivo precio de venta. El listado debe estar ordenado ascendentemente por el número de artículo.

```
SELECT DISTINCT Nart AS Numero_Art, Pv.Nprov AS Numero_prov,
Nombre, Precio_Venta
FROM Proveedores Pd, Provee Pv
WHERE Pd.Nprov =Pv.Nprov
ORDER BY Nart
```

Nota: se puede observar la utilización de alias tanto en el nombre de tablas como en el de columnas.

Resultado

numero_Art	numero_prov	nombre	precio_venta
100	200	Grasano	4
101	200	Grasano	1,1
101	201	Atomo	1
102	201	Atomo	0,5
102	202	VEA	1

Expresiones

Funciones Agregadas

- SQL proporciona operadores agregados que toman el nombre de una columna como argumento. El valor del operador agregado se calcula sobre todos los valores de la columna especificada en la tabla completa. Estas funciones se pueden combinar con la cláusula GROUP BY que se verá mas adelante.

- Si no se quieren eliminar los repetidos para los cálculos se debe utilizar la cláusula distinct (ver ejemplo función COUNT).

Las funciones agregadas son:

- AVG (Calcula el promedio)
- COUNT (Cuenta la cantidad de registros)
- SUM (Suma los valores de una columna)
- MIN (Calcula el mínimo valor de una columna)
- MAX (Calcula el máximo valor de una columna)

AVG

Devuelve el promedio de los valores de una expresión de columna.

Ejemplo:

```
SELECT AVG(Precio) as promedio FROM articulos
```

Devolverá el promedio de precios de todos los artículos.

Resultado

promedio
2,5

SUM

Devuelve la suma total de los valores de una expresión de columna o campo numérico.

Ejemplo:

SELECT SUM(cant) as cantidad FROM articulos

Devolverá la cantidad total de artículos.

Resultado:

cantidad
73

COUNT

Simplemente cuenta el número de registros. No importan los valores que están almacenados en los registros.

Ejemplo 1:

SELECT COUNT(Nart) as cant_articulos FROM Provee

Devolverá la cantidad de artículos diferentes.

Resultado:

cant_articulos
5

Ejemplo 2: si no se quieren contar repetidos:

SELECT COUNT(DISTINCT Nart) as cant_articulos FROM Provee

Devolverá la cantidad de artículos diferentes.

Resultado:

cant_articulos
3

MAX

Devuelve el valor más alto de los contenidos en una expresión de columna.

Ejemplo:

SELECT MAX(Precio) as precio_maximo FROM articulos

Devolverá el precio del artículo más caro.

Resultado

precio_maximo
5,2

MIN

Devuelve el valor más bajo de los contenidos en una expresión de columna.

Ejemplo:

SELECT MIN(Precio) as precio_minimo FROM articulos

Devolverá el precio del artículo más barato.

Resultado

precio_minimo
1,1

Cláusula GROUP BY

La cláusula GROUP BY especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo.
Su sintaxis:

[GROUP BY *expr_columna_group1*
[,*expr_columna_group2*] [,...]]

Donde:

expr_columna_groupN deben coincidir con las expresión de columna utilizada en la cláusula SELECT.

Ejemplo

1) Retornar un listado con los números de artículos y el precio promedio de venta de los proveedores para cada artículo.

```
SELECT Nart, AVG(precio_venta) as promedio_p_v
FROM Provee
GROUP BY Nart;
```

Resultado

Nart	promedio_p_v
100	4
101	1.05
102	0.75

Cláusula Having

La cláusula HAVING trabaja muy similarmente a la cláusula WHERE, y se utiliza para considerar sólo aquellos grupos que satisfagan la condición dada en la cláusula HAVING.

Su sintaxis es:

HAVING *condicionHaving*

Donde *condicionHaving* es la condición que deben cumplir los grupos. Las expresiones permitidas en la cláusula HAVING deben involucrar funciones agregadas. Cada expresión que utilice sólo atributos planos deberá ubicarse en la cláusula WHERE. Por otro lado, toda expresión que involucre funciones agregadas debe aparecer en la cláusula HAVING.

Ejemplo

Retornar un listado con los números de artículos y el precio promedio de venta de los artículos cuyo precio promedio de venta de los proveedores es menor a $2.

```
SELECT Nart, AVG(precio_venta)
FROM Provee
GROUP BY Nart
Having AVG(precio_venta) < 2;
```

Resultado

Nart	avg
101	1.05
102	0.75

UNION

El operador UNION combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de todos los registros devueltos en ambas sentencias. Por defecto, los registros repetidos se omiten. Para no quitarlos se empleará la palabra ALL. Su sintaxis es:

*sentencia*SELECT1 UNION [ALL] *sentencia*SELECT2

Cuando se utilice el operador UNION, la lista de selección para cada sentencia SELECT debe tener el mismo número de expresiones de columnas con el mismo tipo de datos y en el mismo orden (propiedad de unión compatible).

Si utilizamos la opción ALL nos devolverá los repetidos si los hay.

Ejemplo

Devolver el nombre y dirección de todas las personas o empresas del negocio (clientes mas proveedores)

```
SELECT nombre, dir FROM clientes
UNION
SELECT nombre, dir FROM proveedores
```

Resultado

nombre	dir
Pedro	Bs As 90
Juan	Jujuy 800
Maria	Cordoba 24
Grasano	Bs As 100
Atomo	Lima 85
VEA	Alvear 455

INTERSECT

El operador INTERSECT combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de los registros que se encuentran en ambas sentencias al mismo tiempo. Su sintaxis es:

*sentencia*SELECT1 INTERSECT *sentencia*SELECT2

Cuando se utilice el operador INTERSECT, la lista de selección para cada sentencia SELECT debe tener el mismo número de expresiones de columnas con el mismo tipo de datos y en el mismo orden (propiedad de unión compatible).

EXCEPT(Resta)

El operador EXCEPT combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de los registros que se encuentran en la primer sentencia y no en la segunda sentencia. Su sintaxis es:

*sentencia*SELECT1 EXCEPT *sentencia*SELECT2

Cuando se utilice el operador EXCEPT, la lista de selección para cada sentencia SELECT debe tener el mismo número de expresiones de columnas con el mismo tipo de datos y en el mismo orden (propiedad de unión compatible).

Comando INSERT

Este comando se utiliza para agregar uno o mas registros (filas). Esto comando se puede utilizar para efectuar dos tipos de operaciones: a) Insertar un único registro ó b) Insertar en una tabla los registros contenidos en otra.

Comando INSERT para un registro:

INSERT INTO *destino* [(*campo1*[, *campo2*[, ...]])]VALUES (*valor1*[, *valor2*[, ...])

Comando INSERT para insertar mas de un registro:

INSERT INTO *destino* [(*campo1*[, *campo2*[, ...]])] *selección*

Argumento	Descripción
Destino	El nombre de la tabla donde se van a añadir registros.
CampoN	Los nombres de los campos donde se van a añadir los datos, si no se ponen se asume que son todos atributos de la tabla destino.
Selección	Es una expresión de selección que debe tener el mismo número de columnas y tipos que las columnas a insertar. Los registros que de cómo resultado la expresión SELECT serán agregados en la tabla destino (ver comando SELECT). Esto permite insertar en una tabla varios registros.
ValorN	Los valores que se van a insertar en los campos específicos del nuevo registro. Cada valor se inserta en el campo que corresponde a la posición del valor en la lista: valor1 se inserta en campo1 del nuevo registro, valor2 dentro de campo2, y así sucesivamente. Debe separar los valores con una coma y escribir los campos de texto entre comillas (' ')y las fechas indicarlas en formato dd/mm/aa y entre comillas o N.

Ejemplo

1) INSERT INTO articulos
(Nart,Descripcion,precio,Stock_max,Stock_min)
VALUES (5, 'Vino x1L', 1.5,90,20);

En este ejemplo se agrega un nuevo registro a la tabla artículos, pero en el campo cantidad no se pone ningún valor (ver definición tabla artículos)

2) INSERT INTO proveedores SELECT * FROM clientes

En este ejemplo se agrega a la tabla proveedores todos los clientes

Comando UPDATE

Para cambiar uno o más valores de campos de registros en una tabla, se utiliza el comando UPDATE.

Su sintaxis es:

UPDATE tabla SET campo1 = valor1 [, campo2 = valor2 [, ...]]
WHERE condición;

Argumento	**Descripción**
Tabla	Nombre de la tabla cuyos datos desea modificar.
CampoN	Nombre del campo cuyo valor se actualizará.
ValorN	Expresión cuyo valor tomará el campoN. La expresión debe ser del tipo del campo.
Condición	Una expresión que determina qué registros se actualizarán. Sólo se actualizan los registros que satisfacen la expresión.

Ejemplos

1)
UPDATE articulos SET precio = precio * 1.2;

Esta instrucción incrementa los precios de todos los artículos en un 20 por ciento.

2)
UPDATE articulos SET precio = precio * 1.2
WHERE precio <2;

La ejecución de este comando incrementa los precios de los artículos cuyo precio es menor a $2, en un 20 por ciento.

Comando DELETE

El comando DELETE se utiliza para borrar uno o varios registros de una tabla particular.

Su sintaxis es:

DELETE FROM *tabla* [WHERE *condición*];

Argumento	**Descripción**
Tabla	Nombre de la tabla cuyos registros se van a eliminar.
Condición	Expresión que determina qué registros se van a eliminar.

Nota:
Si se omite la cláusula WHERE se eliminan todos los registros de la tabla.

Ejemplos

1)
DELETE FROM proveedores;

La ejecución de este comando producirá que la tabla proveedores quede vacía.

2)
DELETE FROM proveedores
WHERE Nprov=200;

Este comando elimina el registro correspondiente al proveedor número 200.

Reuniones (JOIN)

Se utilizan dentro de la cláusula FROM y su sintaxis es:

tabla1 [NATURAL] { {LEFT|RIGHT|FULL} OUTER | INNER} JOIN *tabla2* [ON *condicionJoin*][using (*columna1[, columna2[, ...]])]

Donde *tablaN* son las tablas involucradas en la reunión , *condicionJoin* es la condición(θ) del join y *columnaN* son las columnas involucradas en el join.

Su comportamiento es similar al visto en el álgebra relacional.

Ejemplos

- Obtener un listado de **todos** los proveedores y los artículos que provee, si un proveedor no provee ningún artículo, debe ser listado.

```
SELECT DISTINCT Nart, Pv.Nprov, Nombre, Precio_Venta FROM
 proveedores pv NATURAL LEFT OUTER JOIN provee
```

O

```
SELECT DISTINCT Nart, Pv.Nprov, Nombre, Precio_Venta  FROM
 proveedores pv LEFT OUTER JOIN provee P ON (Pv.Nprov = P.Nprov)
```

O

```
SELECT DISTINCT Nart, Pv.Nprov, Nombre, Precio_Venta  FROM
 proveedores pv LEFT OUTER JOIN provee USING (Nprov)
```

Traducción del álgebra relacional a SQL

A continuación se presentan reglas utilizadas en la traducción de formulas escritas con los operadores básicos del álgebra relacional a instrucciones en el lenguaje SQL.

Álgebra relacional	**SQL**
$\pi_{a,b}\ (\sigma_{condicion}\ (r1\ X\ r2))$	SELECT a,b FROM R1, R2 WHERE condición
$\pi_{a,b}\ (r1) \cup\ \pi_{a,b}\ (r2)$	SELECT a,b FROM R1 UNION SELECT a,b FROM R2
$\pi_{a,b}\ (r1)$ - $\pi_{a,b}\ (r2)$	SELECT a,b FROM R1 EXCEPT SELECT a,b FROM R2

Nota: debería ser SELECT DISTINCT en vez de SELECT solamente, debido a que en el álgebra relacional las tablas son conjuntos

Traducción del álgebra relacional a SQL (sigue)

A continuación se presentan reglas utilizadas en la traducción de formulas escritas con los operadores **no** básicos del álgebra relacional a instrucciones en el lenguaje SQL.

Álgebra relacional	**SQL**
r1 ⋈ r2	SELECT * FROM r1 NATURAL JOIN r2
$\pi_{a,b}(r1) \cap \pi_{a,b}(r2)$	SELECT a,b FROM R1 INTERSECT SELECT a,b FROM r2
r1 ⟕ r2	SELECT * FROM r1 NATURAL LEFT OUTER JOIN r2
……	……

Ejemplos

1) Lista de artículos a reponer:

En el álgebra relacional:

$\sigma(_{stock\_min > cant}$ (articulos)

en SQL

SELECT * FROM articulos WHERE stock_min>cant;

2) Listado de artículos con su código, descripción y precio:

En el álgebra relacional:

$\pi_{NArt,Descripción,Precio}$ (articulos)

en SQL:

SELECT Nart, desc, precio FROM articulos;

Ejemplos(sigue)

3) Listado de artículos con indicación del proveedor que lo suministra y a que precio, el listado debe contener las siguientes columnas(Nart, Nprov, Nombre, Precio_venta)

En el álgebra relacional:

$$\pi_{\text{Nart,Nprov, Nombre, Precio\_Venta}}(\text{proveedor } |X| \text{ provee})$$

en SQL

```
SELECT DISTINCT Nart, Pv.Nprov, Nombre, Precio_Venta
 FROM proveedores Pd, provee  Pv   WHERE Pd.Nprov =Pv.Nprov
```

O más eficiente es realizarla utilizando el operador JOIN de SQL

```
SELECT DISTINCT Nart, Pv.Nprov, Nombre, Precio_Venta
      FROM proveedores pv NATURAL JOIN provee
```